FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Rozpoznávání objektů hlubokou sítí pro robota NAO
Student:	Martin Kostelanský
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Proveďte rešerši dostupných předučených modelů pro rozpoznávání objektů hlubokými neuronovými sítěmi, vyhodnoťte je z hlediska přesnosti klasifikace objektů a použitelností pro aplikace robota NAO. Dále prozkoumejte možnost předučené modely doučovat rozpoznávání nových objektů a proveďte potřebné experimenty. Vybrané modely hlubokých neuronových sítí integrujte spolu s programovým vybavením robota NAO a vytvořte funkční aplikaci, kde bude robot NAO reagovat na předměty ve scéně před sebou. Rozsah práce upřesněte po dohodě s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 9. ledna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Rozpoznávání objektů hlubokou sítí pro robota NAO

Martin Kostelanský

Katedra teoretické informatiky
Vedúci práce: Ing. Miroslav Skrbek, Ph.D.

10. mája 2018

Poďakovanie

Chcel by som poďakovať rodine a priateľom za dôveru a podporu, ktorá ma sprevádzala počas môjho štúdia. Taktiež by som sa chcel poďakovať aj vedúcemu mojej bakalárskej práce, bez ktorého odbornej pomoci a nadšenia pre danú oblasť by táto práca nikdy nevznikla.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb, autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k používaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo používať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela, ale len pre nezárobkové účely. Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Prahe 10. mája 2018 .....

Odkaz na túto prácu

Kostelanský, Martin. *Rozpoznávání objektů hlubokou sítí pro robota NAO.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

V mojej práci skúmam strojové videnie a možnosti rozpoznávania objektov. Používam najmodernejšie hlboké konvolučné neurónové siete vo frameworku Keras. Skúmal som ich architektúru a možnosti dotrénovania. V rámci experimentov som vytvoril novú sieť kombináciou dvoch typov sietí a ukázalo sa, že takto vytvorená sieť je úspešnejšia v klasifikácii ako tie, z ktorých bola vytvorená. Vybrané siete som vytrénoval na rozpoznávanie priestorov Fakulty informačných technológií ČVUT v Prahe. Najvhodnejšie z nich som implementoval v aplikácií pre robota NAO, pomocou ktorej dokáže určiť v akom priestore sa nachádza na fakulte s presnosťou $93,48\%$ a s presnosťou $76,09\%$ dokáže určiť, v ktorej z budov sa práve stojí.

Klúčové slová rozpoznávanie obrazu, NAO, konvolučná neurónová sieť, hlboké učenie, predtrénovaný model, Python, Keras

Abstract

In my work, I researched machine vision and object recognition capabilities. I used the newest deep convolutional neural networks in framewok Keras. I examined their architecture and possibilities of their fine-tuning. In my experiments, I created new network combining two types of networks. Results showed that this network is more successful in classification than the original ones. I trained selected networks for space recognition at Faculty of information technology CTU in Prague. The best networks were implemented in application for robot NAO, by which it can determine with 93.48% accuracy where at the faculty is he located and with 76.09% accuracy in which of the buildings is he standing.

Keywords image recognition, NAO, convolutional neural network, deep learning, pretrained model, Python, Keras

Obsah

Úvod **1**

1 Cieľ práce **3**

2 Konvolučné neurónové siete **5**
2.1 Architektúra 5
2.2 Trénovanie 7

3 Predtrénované modely **11**
3.1 VGG16 11
3.2 ResNet50 14
3.3 MobileNet 17

4 Výpočtové prostredie **19**
4.1 Google Colaboratory 19

5 Robot NAO **21**
5.1 NAOqi framework 21

6 Keras **23**

7 Testovanie a trénovanie vybraných modelov **25**
7.1 Testovanie predtrénovaných modelov 25
7.2 Testovanie presnosti modelov vzhľadom na hĺbku dotrénovania 25
7.3 Trénovanie na vytvorenom datasete školy 30

8 Implementácia **35**
8.1 Výber modelov 35
8.2 Pohyb a odfotenie priestoru 36
8.3 Klasifikácia 36
8.4 Interakcia 36

8.5 Architektúra 39
8.6 Príklad použitia 39

9 Výsledky a diskusia **41**

Záver **43**

Literatúra **45**

A Zoznam použitých skratiek **47**

B Obsah priloženého DVD **49**

Zoznam obrázkov

2.1 Architektúra konvolučných neurónových sietí 6
2.2 Porovnanie priemerného a maximálneho spájania 7

3.1 Architektúra VGG16 13
3.2 Stavebný blok reziduálneho učenia 15
3.3 Architektúra ResNet50 16
3.4 Rozloženie klasickej konvolúcie na hĺbkovo oddeliteľnú 18

4.1 Príklad práce v Goole Colaboratory 20

5.1 Robot NAO 22

6.1 Príklad použitia frameworku Keras 23

7.1 Trénovanie modelu vo frameworku Keras 26
7.2 Vytvorenie použitého modelu VGG16 vo frameworku Keras 26
7.3 Vytvorenie použitého modelu ResNet50 vo frameworku Keras . . . 27
7.4 Trénovacia strata VGG16 v priebehu trénovania 27
7.5 Validačná strata VGG16 v priebehu trénovania 28
7.6 Trénovacia strata ResNet50 v priebehu trénovania 28
7.7 Validačná strata ResNet50 v priebehu trénovania 29
7.8 Porovnanie VGG16 a ResNet50 29
7.9 Vytvorenie použitého modelu ResNet50_B vo frameworku Keras . 31
7.10 Trénovacia strata pri klasifikácii budov 32
7.11 Validačná strata pri klasifikácii budov 32
7.12 Trénovacia strata pri klasifikácii priestorov 33
7.13 Validačná strata pri klasifikácii priestorov 33

8.1 Zachytenie fotografií a ovládanie pohybu cez NAOqi API 37
8.2 Použitie NAOqi na komunikáciu a reagovanie na stlačenie tlačidiel 38
8.3 Architektúra aplikácie pre robota NAO 39

8.4 Spustenie aplikácie pre robota NAO 40

Zoznam tabuliek

3.1 Porovnanie predtrénovaných modelov 11

4.1 Porovnanie doby behu na CPU versus GPU 20

7.1 Výsledky testovania predtrénovaných modelov 25
7.2 Počet fotografií v jednotlivých kategóriách 30
7.3 Počet fotografií v trénovacej a validačnej množine datasetu priestorov. 30
7.4 Počet fotografií v trénovacej a validačnej množine datasetu budov. 30

8.1 Porovnanie veľkostí modelov 36

9.1 Výsledky testovania pre miesta 41
9.2 Výsledky testovania pre budovy 41
9.3 Výsledky pre všetky kategórie 42

Úvod

Jedným z najprogresívnejších odvetví informačných technológií súčasnosti je nepopierateľne strojové učenie. S nárastom výkonnosti počítačov za posledné desiatky rokov, sa stalo využitie neurónových sietí výpočetne možné v rozumnom čase. Vďaka tomu sa umelá inteligencia stáva súčasťou nášho každodenného života, nielen vo forme doporučovacích systémov a hlasových asistentov, ale pomaly si nachádza miesto v mnohých ďalších odvetviach. Inteligentné systémy sa začínajú implementovať do strojov, ktoré sa vďaka nim stávajú plne autonómne. Rozpoznávanie obrazu patrí k najväčším problémom, ktoré je potrebné vyriešiť pri integrácii strojov do reálneho sveta.

Táto práca slúži ako prvotné skúmanie možnosti využitia a implementácie hlbokých konvolučných sietí na robotov na Katedre číslicového návrhu Fakulty informačných technológií ČVUT v Prahe. Pri inteligentých aplikáciách je schopnosť autonómneho určenia polohy kľúčová. Jedným z hlavných cieľov tejto práce, okrem preskúmania možností práce s konvolučnými neurónovými sieťami a ich dotrénovania, je práve riešenie tohto problému, vytvorenie aplikácie na robota NAO pomocou ktorej bude schopný určiť v akom priestore a v akej budove sa práve nachádza.

V úvode mojej práci sa zaoberám princípmi architektúry hlbokých konvolučných neurónových sietí a ich trénovaním, za čím nasleduje rozbor niekoľkých sietí používaných v praxi, ktorých autormi sú popredné technologické spoločnosti alebo univerzity svetového formátu. Následne predstavujem nástroje, pomocou ktorých som s týmito sieťami pracoval a trénoval ich, ako aj výsledky týchto experimentov. Na záver predstavujem robota NAO a postup pri vývoji finálnej aplikácie. Prácu uzatvára analaýza výsledkov a vyhodnotenie.

Kapitola 1

Cieľ práce

Cieľom mojej práce je preskúmať dostupné predtrénované modely vo frameworku Keras. Následne porovnať ich vnútornú štruktúru a otestovať ich klasifikačnú presnosť a schopnosť dotrénovania na nové dáta. Na základe výsledkov mojich experimentov vyberiem najvhodnejšie modely pre implementáciu na robota NAO. Následne bude mojou úlohou dané modely doučiť na mnou vytvorenom datasete obsahujúcom fotografie Fakulty informačných technológií ČVUT v Prahe. Najúspešnejšíie modely budú použité v aplikácii pre robota NAO pomocou ktorej bude schopný určiť v akom priestore a v akej budove sa práve nachádza.

Kapitola 2

Konvolučné neurónové siete

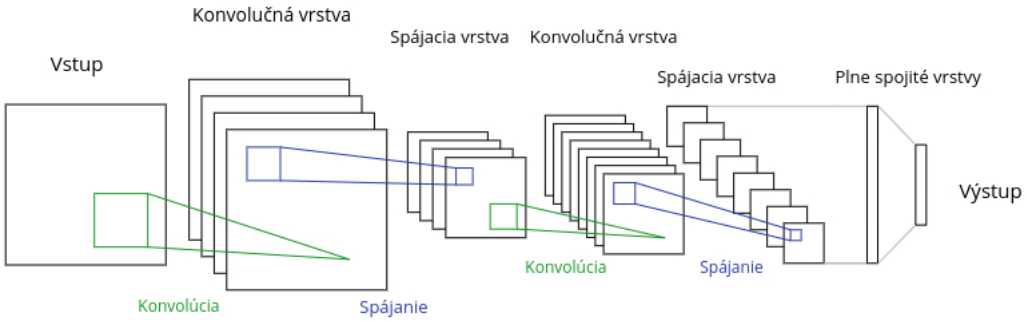
Klasifikácia obrazu, krorá môže byť definovaná ako úloha kategorizovať obraz do jednej z predurčených kategórií, patrí k základným problémom v strojovom videní. To vytvára ďalšie typy úloh spojených so strojovým videním ako lokalizácia, detekcia a segmentácia. V súčasnosti sú konvolučné neurónové siete dominujúcou architektúrou pre rozpoznávanie obrazu, klasifikáciu a detekciu.

Napriek ich skorším úspechom, do popredia sa dostali až s použitím hlbokého učenia podporeného výkonnými grafickými kartami, veľkými datasetmi a vylepšenými algoritmami. Veľmi dôležitý faktor, ktorý zohral úlohu v zvýšení záujmu o hlboké konvolučné neurónové siete, bola súťaž ILSVRC 2012 (ImageNet Large Scale Visual Recognition Challenge), v ktorej víťaz použil hlbokú konvolučnú sieť na klasifikáciu približne 1,2 milióna obrázkov do tisícich tried[1].

2.1 Architektúra

Konvolučné neurónové siete sú dopredné siete, v ktorých prúd informácií tečie jednosmerne, zo vstupu na výstup. Tak ako umelé neurónové siete, aj CNNs majú biologický základ. Predlohou ich architektúry je vizuálna kôra v mozgu, ktorá sa skladá zo striedajúcich sa vrstiev jednoduchých a komplexných buniek.

Architektúra CNNs existuje v niekoľkých variáciách, vo všeobecnosti sa však skladajú z konvolučných a spájacích vrstiev, ktoré sú zoskupené do modulov. Po týchto moduloch nasleduje jeden alebo viac plne spojítých vrstiev, aké poznáme z klasických dopredných neurónových sietí. Moduly sú často naskladané na seba a vytvárajú tak hlboký model. Vstupom do siete je klasifikovaný obraz, za ktorým nasleduje niekoľko vrstiev konvolúcie a spájania. Potom reprezentácie z týchto vrstiev naplnia jednu alebo viac plne spojitých vrstiev. Výstupom z poslednej plne spojitej vrstvy je predikcia klasifikácie obrazu na vstupe[1]. Architektúru popisuje ilustrácia 2.1.



Obr. 2.1: Architektúra konvolučných neurónových sietí

2.1.1 Konvolučné vrstvy

Konvolučné vrstvy slúžia ako nástroj extrakcie a jej parametre pozostávajú z množiny učiteľných filtrov. Priestorovo je každý z nich malý, ale posúvajú sa naprieč celým vstupom. Každý neurón v CONVL má receptorové pole pozostávajúce z množiny neurónov v prechádzajúcej vrstve, ktoré sú s ním prepojené pomocou filtra. Vstupy konvolujú pomocou učiteľných váh filtra a nelineárnej aktivačnej funkcie. Pre neurón na pozícii i, j v skrytej vrstve platí

$$Y_{i,j} = \sigma(b + \sum_{k=0}^{x}\sum_{l=0}^{y} w_{k,l}a_{i+k,j+l}), \tag{2.1}$$

kde σ predstavuje nelineárnu aktivačnú funkciu, b zdieľaný posun, $w_{l,m}$ zdieľanú váhu a $a_{j+l,k+m}$ vstupnú hodnotu neurónu na pozícii $j+l, k+m$. Kompletná konvolučná vrstva pozostáva z niekoľkých filtrov na každej úrovni[2].

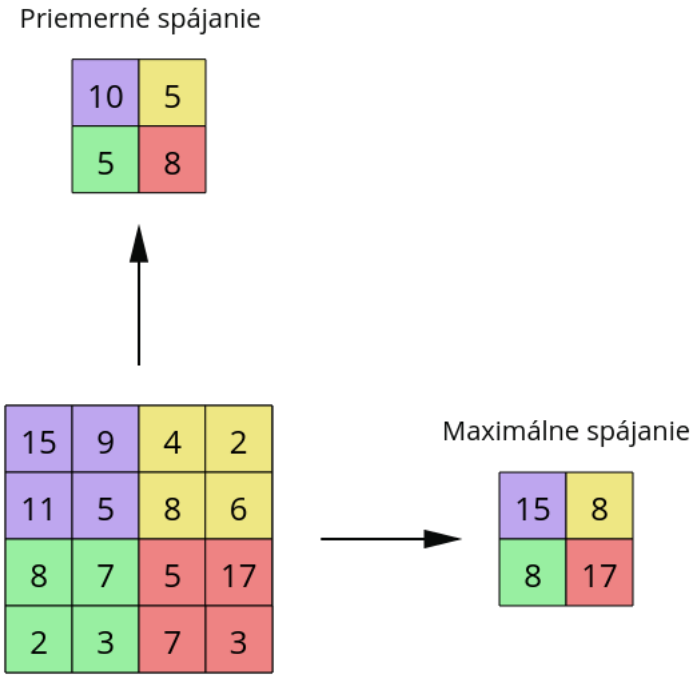
2.1.2 Spájacie vrstvy

Spájacie vrstvy obvykle nasledujú hneď po konvolučných. Slúžia na redukovanie priestorového rozlíšenia máp funkcií a potlačenie šumu na vstupe. Spočiatku sa bežne používalo priemerné spájanie, ktoré propagovalo priemernú hodnotu malej oblasti do ďalšej vrstvy. V posledných rokoch sa dostala do popredia metóda maximálneho spájania, ktorá propaguje do ďalšej vrstvy prvok s najäčšou hodnotou v receptorovom poli

$$Y_{i,j} = \max_{(p,q)\in\Re_{i,j}}(x_{p,q}), \tag{2.2}$$

kde $Y_{i,j}$, označuje výstup spájacej operácie a $x_{p,q}$ označuje prvok na pozícii (p, q) v receptorovom poli $\Re_{i,j}$. Obrázok 2.2 ilustruje rozdiel medzi priemerným

a maximálnym spájaním. Metóda priemerného spájania vyberá z receptorového pola, v tomto prípade o veľkosti 2×2 priemer, pričom metóda maximálneho spájania vyberá najväčší prvok[1].



Obr. 2.2: Porovnanie priemerného a maximálneho spájania

2.1.3 Plne spojité vrstvy

Obvykle je na sebe nakopených niekoľko konvolučných a spájacích vrstiev, aby extrahovali zo vstupu viac abstraktnú reprezentáciu. Za nimi nasledujú plne spojité vrstvy, ktoré slúžia na vysokoúrovňové myslenie. Pri klasifikačných problémoch je štandardné použitie soft-max[3] operátora na vrchu hlbokej konvolučnej neurónovej siete. Použitie tohto operátora viedlo k zvýšeniu klasifikačnej presnosti[1].

2.2 Trénovanie

CNNs a ANNs používajú vo všeobecnosti učiace algoritmy na úpravu voľných parametrov (váh a posunov), aby dosiahli požadovaný výstup. Najbežnejšia

metóda používaná na tento účel je algoritmus spätného šírenia, ktorý vypočíta gradient cieľa (tiež nazývaný ako cena, strata), aby určil ako upraviť parametre siete za účelom minimalizovať chyby, ktoré ovplyvňujú výkonnosť. Bežný problém s trénovaním CNNs a špeciálne DCNNs je pretrénovanie, čo je slabý výkon po dotrénovaní na malých alebo dokonca aj veľkých dátach. Ovplyvňuje to schopnosť modelu generalizovať na nových dátach[1].

2.2.1 Stochastický gradientný zostup

Štandardný gradientný zostup aktualizuje parametere θ cieľa $J(\theta)$ ako

$$\theta = \theta - \alpha \nabla_\theta E[J(\theta)], \tag{2.3}$$

kde sa očakávanie približuje vyhodnotením ceny a gradientu naprieč celou trénovacou množinou. SGD jednoducho odstráni očakávanie v aktualizácii a vypočíta gradient parametrov pomocou jedného alebo niekoľkých tréningových príkladov,

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \tag{2.4}$$

s párom $(x^{(i)}, y^{(i)})$ z trénovacieho setu.

Vo všeobecnosti je každá aktualizácia parametrov v SGD počítaná po niekoľkých trénovacích príkladoch, alebo po malej skupine trénovacích dát. Dôvod je dvojaký, za prvé to znižuje rozdiely v aktualizácii parametrov a môže viesť k stabilnejšej konvergencii, po druhé, to umožnuje, aby boli pri výpočte použité vysoko optimalizované maticové operácie, ktoré by mali byť použité vo vektorizovanom výpočte cieľa a gradientu. Typická veľkosť mini skupiny trénovacích dát je 256, avšak optimálna veľkosť tejto skupiny sa môže líšiť pre rôzne aplikácie a architektúry.

V SGD je stupeň učenia α zvyčajne oveľa menší ako zodpovedajúci stupeň pri skupinovom gradientnom zostupe, pretože v aktualizácii je obeľa väčší rozptyl. Výber správneho stupňa učenia a plánu, zmena stupňa učenia v priebehu tréningu, môže byť veľmi zložité. Jednou zo štandartných metód, ktorá funguje dobre v praxi, je použiť dostatočne malú konštantú úroveň učenia, ktorá stabilne konverguje v počiatočných epochách tréningu a následne znížiť stupeň učenia ako sa konvergencia spomaľuje[4].

2.2.2 Momentum

Ak má cieľ tvar dlhej plytkej rokliny so strmými stenami na stranách vedúcej k optimu, štandartné SGD bude mať tendenciu oscilovať naprieč roklinou, pretože gradient bude skôr smerovať nadol jednou zo strmých stien ako pozdĺž rokliny smerom k optimu. Ciele hlbokých architektúr majú tento tvar blízko lokálneho optima a preto štandartné SGD môže viesť k veľmi pomalej konvergencii, najmä po úvodných prudkých ziskoch. Momentum je jednou

z možností, ako rýchlejšie tlačiť cieľ pozdĺž plytkej rokliny. Momentová aktualitácia je daná vzťahom

$$\begin{aligned} v &= \gamma v + \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \\ \theta &= \theta - v. \end{aligned} \tag{2.5}$$

Vo vyššie uvedenej rovnici je v aktuálny vektor rýchlosti, ktorý ma rovnaký rozmer ako vektorový parameter θ. Pri použití momenta môže byť potrebné zvoliť nižší stupeň učenia α, pretože veľkosť gradientu bude väčšia. Parameter $\gamma \in \langle 0, 1\rangle$ určuje, koľko predchádzajucích iterácií gradientu je začlenejnej do aktuálnej aktualizácie. Vo všeobecnosti je γ nastavená na $0,5$, kým sa počiatočné učenie stabilizuje a potom je zvýšená na $0,9$ alebo vyššie[4].

Kapitola 3

Predtrénované modely

Predtrénované modely sú také, ktorých váhy boli predtrénované na obsiahlych datasetoch. Modely, ktoré som pužil sú voľne dostupné vo frameworku Keras a všetky boli predtrénované na obrazovej databáze ImageNet, ktorá obsahuje približne 15 miliónov obrázkov rozdelených do tisícky kategórií[5]. Tabuľka 3.1 popisuje jednotlivé modely. Parametre Top-1 a Top-5 popisujú presnosti modelov na ImageNet validačnom datasete, pri kategórii Top-1 je skúmané, či model správne klasifikoval vstup. Pri kategórií Top-5 je braných do úvahy 5 predikcií s najvyššou pravdepodobnosťou. K ďalšiemu rozboru som zvolil tri modely: VGG16, ResNet50 a MobileNet.

Model	Veľkosť [MB]	Top-1	Top-5	Počet parametrov
Xception	88	0,790	0,945	22 910 480
VGG16	528	0,715	0,901	138 357 654
VGG19	549	0,727	0,910	143 667 240
ResNet50	99	0,759	0,929	25 636 712
InceptionV3	92	0,788	0,944	23 851 784
InceptionResNet V2	215	0,804	0,953	55 873 736
MobileNet	17	0,665	0,871	4 253 864
DenseNet121	33	0,745	0,918	8 062 504
DenseNet169	57	0,759	0,928	12 307 880
DenseNet201	80	0,770	0,933	20 242 984

Tabuľka 3.1: Porovnanie predtrénovaných modelov[6]

3.1 VGG16

Táto konvolučná neurónova sieť bola vytvorená skupinou vizuálnej geometrie na Univerzite v Oxforde počas skúmania vplyvu hĺbky siete na úspešnosť v širokom rozpoznávaní objektov[7]. S touto sieťou sa v roku 2014 umiestnili

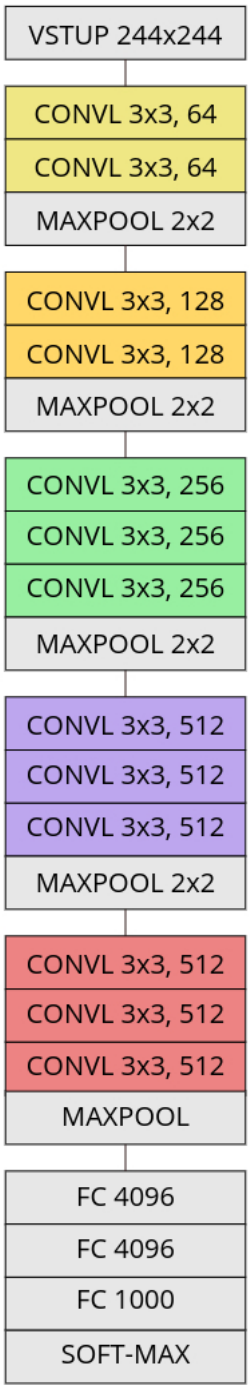
na prvom a druhom mieste v kategórách lokalizácia a klasifikácia v súťaži ILSVRC[8].

3.1.1 Architektúra

Počas tréningu bol vstup do CNN zafixovaný na rozmery 224×224 pixelov. Obrázok pokračuje cez sériu konvolučných vrstiev, kde je používané veľmi malé receptorové pole o veľkosti 3×3 pixely s posunom jeden pixel. Priestorové spájanie je zabezpečené piatimi spájacími vrstvami, v ktorých je použitá metóda maximálneho spájania. Tieto vrstvy sa nachádzajú medzi blokmi konvolučných vrstiev. Počiatočná šírka konvolučnej vrstvy je pomerne malá, začína na 64 a posupne sa zvyšuje násobením dvomi, pokiaľ nedosiahne 512. Zvyšovanie šírky prebieha medzi jednotlivými blokmi. Maximálne spájanie je aplikované na pixelové pole s veľkosťou 2×2, s rozostupom 2 pixely. Za týmito vrstvami nasledujú tri plne spojité vrstvy. Prvé dve majú po 4096 neurónov a za nimi nasleduje vrstva, ktorá zaraďuje vstup do jednej z tisícich kategórií. Za ňou je finálna vrstva zabezpečujúca normalizáciu pomocou soft-max funkcie. Obrázok 3.1 detailne popisuje sled jednotlivých vrstiev, aj s veľkosťami receptorových polí, šírkami konvolučných vrstiev, alebo počtom neurónov v plne spojitej vrstve[7].

3.1.2 Trénovanie

Natrénovanie prebiehalo pomocou SGD podporeného použitím malých skupín vzoriek a momenta. Veľkosť skupiny bola 256 a momentum $0,9$. Stupeň učenia bol spočiatku nastavený na 10^{-2},a potom bol znižovaný delením 10, keď sa prestala zlepšovať presnosť na validačnom sete. Celkovo bol stupeň učenia znížený tri krát a učenie bolo zastavené po 37×10^4 iteráciách, 74 epochách[7].



Obr. 3.1: Architektúra VGG16

3.2 ResNet50

Resnet50 bola vyvinutá výskumným tímom spoločnosti Microsoft a v roku 2015 táto sieť zvíťazila v súťaži ILSVRC v kategóriách klasifikácie a detekcie[9]. V ich práci reformulovali vrstvy ako učiace sa reziduálne funkcie s referenciou na zdrojovú vrstvu. Ukázali ľahkosť tréningu reziduálnych konvolučných neurónových sietí a zvýšenie klasifikačnej presnosti s rastúcou hĺbkou[10].

3.2.1 Hlboké reziduálne učenie

3.2.1.1 Reziduálne učenie

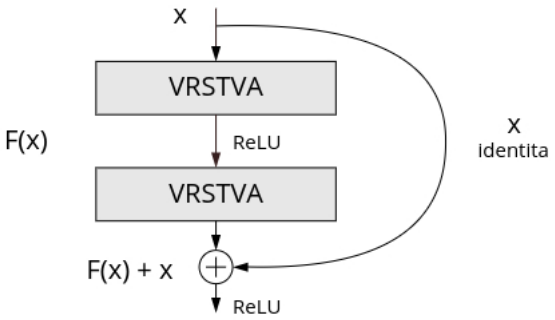
Považujme H(x) ako základné mapovanie niekoľkých nakopených vrstiev, kde x predstavuje vstup do prvej z týchto vrstiev. Ak sa predpokladá, že niekoľko nelineárnych vrstiev môže asymptoticky aproximovať komplikované funkcie, môžeme predpokladať, že môžu asymptoticky aproximovať reziduálne funkcie, t.j. $H(x) - x$, za prespokladu že vstup a výstup sú toho istého rozmeru. Namiesto toho, aby sme očakávali, že nakopené vrstvy aproximujú $H(x)$, explicitne necháme tieto vrstvy aproximovať reziduálnu funkciu $F(x) := H(x) - x$. Tým sa pôvodnou funkcou stáva $F(x) + x$. Aj keď obe formy by mali byť asymptoticky aproximovateľné, jednoduchosť učenia sa môže líšiť. Táto reformulácia je motivovaná neintuitívnym problémom degradácie hlbokých sietí. Vzhľadom na to, že vrstvy pridané do tejto siete sú konštruované ako identitné mapovanie, chybovosť by nemala byť väčšia ako jej plytšia verzia. Problém degradácie naznačuje, že môže nastať problém v aproxmovaní identitného mapovania niekoľkými nelienárnymi vrstvami. S refromuláciou reziduálneho učenia, za predpokladu optimálneho mapovania identity, môžeme riadiť váhy niekoľkých nelineárnych vrstiev k nule a pribížiť sa k mapovaniu identity[10].

3.2.1.2 Mapovanie identity podľa skratiek

Reziduálne učenie používame na každých niekoľko vrstiev. Formálne definujeme stavebnú jednotku ako

$$y = F(x, W_i) + x, \tag{3.1}$$

kde x a y sú vstupné a výstupné vektory jednotlivých vrstiev. Funkcia $F(x, W_i)$ reprezentuje reziduálne mapovanie potrebné natrénovať. Napríklad vo figúre 3.2, ktorá ma dve vrstvy, $F = W_2\sigma(W_1x)$ v ktorej σ znázorňuje aktivačnú funkciu ReLU[3] a posuny sú vynechané pre jednoduchosť zápisu. Operácia $F + x$ sa uskutočňuje pomocou prepojenia na skratku a rozumným sčítaním prvkov. Po sčítaní používame druhú nelineárnu funkciu[10].



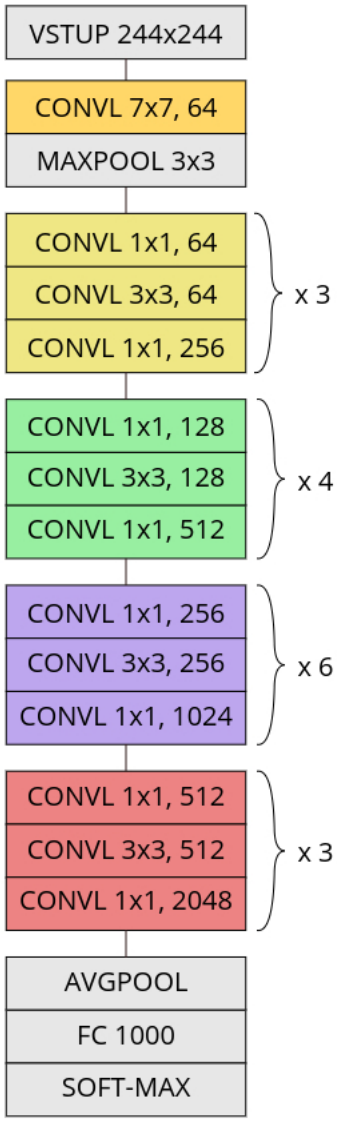
Obr. 3.2: Stavebný blok reziduálneho učenia

3.2.2 Architektúra

Základné línie siete sú inšpirované sieťami VGG. Vstup do siete bol nastavený na 244×244 pixelov, za ktorým nasleduje konvolučná vrstva s filtrom o veľkosti 7×7 pixelov a posunom dva pixely. Nasleduje maximálne spájanie s receptorvým poľom o veľkosti 3×3 a posunom 2 pixely. Ďalej sú použité bloky konvolučných vrstiev v ktorých sú použité filtre o veľkosti 1×1 a 3×3. Za konvolučnými vrstvami sa nachádza spájacia, v ktorej je použité globálne priemerné spájanie nasledované plne spojitou vrstvou, v ktorej je použitá funkcia soft-max a slúži na finálnu predikciu. V sieti sú konvolučné vrstvy s filtrom veľkým 3×3 pixely prepojené skratkami, typickými pre reziduálne učenie. Usporiadanie jednotlivých vrstiev je detailnejšie vidieť na obrázku 3.3, ktorý však pre jednoduchosť neuvádza reziduálne prepojenia[10].

3.2.3 Trénovanie

Na trénovanie bol použitý algoritmus SGD s veľkosťou skupín 256 a momentom. Stupeň učenia bol spočiatku nastavený na $0,1$ a bol delený desiatimi ako sa chybovosť znižovala. Momentum bolo nastavené na $0,9$ a trénovanie prebiehalo 60×10^4 iterácií[10].



Obr. 3.3: Architektúra ResNet50

3.3 MobileNet

MobileNet je efektívna neurónová sieť určená pre aplikácie na mobilné a vstavané zariadenia, vyvinutá spoločnosťou Google. Tento model predstavuje úspornú architektúru používajúcu hĺbkovo oddeliteľnú konvolúciu na odľahčenie konvolučných neurónových sietí. Kompromis medzi latenciou a presnosťou je nastavovateľný pomocou dvoch globálnych hyperparametrov[11].

3.3.1 HĺBkovo oddeliteľná konvolúcia

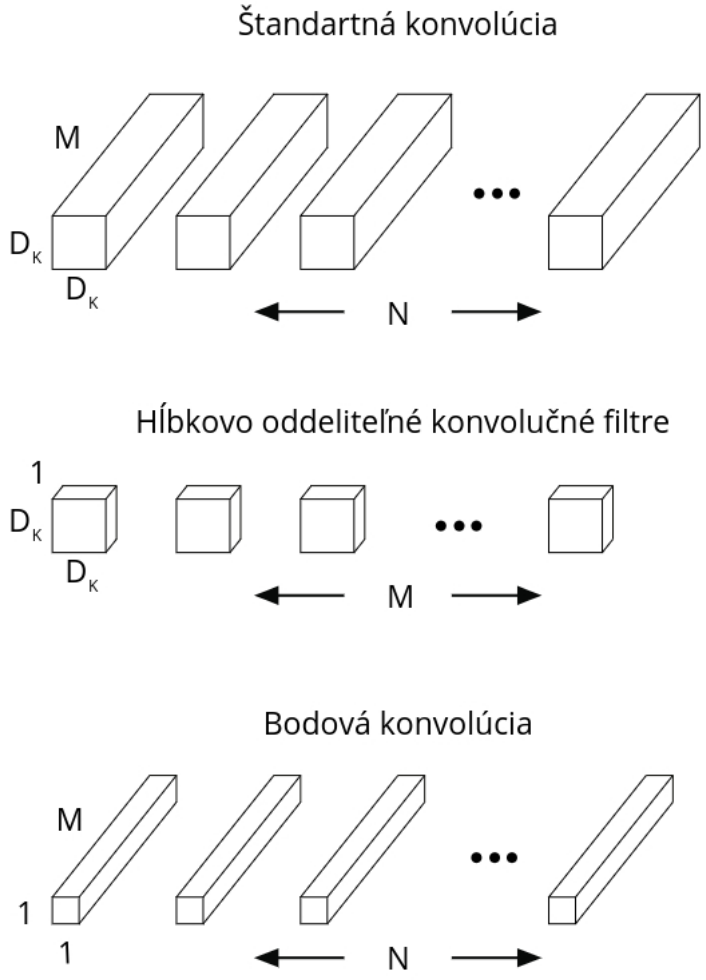
Model MobileNet je založený na hĺbkovo oddeliteľnej konvolúcii, ktorá je forma faktorizovanej konvolúcie. Tá faktorizuje štandartnú konvolúciu na hĺbkovo oddeliteľnú konvolúciu a 1×1 konvolúciu, ktorá sa tiež nazýva bodová. MobileNet pri hĺbkovo oddeliteľnej konvolúvcii aplikuje jeden filter pre každý vstupný kanál. Štandartná konvolúcia filtruje a kombinuje vstupy do nového setu výstupov v jednom kroku. HĺBkovo oddeliteľná konvolúcia rozdelí tento proces do dvoch vrstiev. Jedna slúži na filtrovanie a druhá na kombinovanie. HĺBkovo oddeliteľná konvolúcia aplikuje jeden filter pre každý vstupný kanál. Následne je použitá bodová konvolúcia na vytvorenie lineárnych kombinácii výstupu hĺbkovo oddeliteľnej vrstvy. Obrázok 3.4 ukazuje, ako je štandartná konvolúcia faktorizovaná na hĺbkovo oddeliteľnú a bodovú, kde D_K predstavuje rozmery vstupu, M jeho šírku a N počet vrstiev. Táto faktorizácia má za efekt drastickú redukciu výpočetnej zložitosti a veľkosti modelu[11].

3.3.2 Architektúra

Veľkosť vstupu je zafixovaná na 224×224 pixelov, za ktorým nasleduje klasická konvolúcia, po ktorej prichádza hĺbkovo oddeliteľná. Pri hĺbkovo oddeliteľnej konvolúcii sú použité receptorové polia o veľkosti 3×3 pixely s posunom 1 pixel. Za týmito vrstvami nasleduje priemerné spájanie s receptorovým poľom 7×7 nasledovaným plne spojitou vrstvou a soft-max normalizáciou. Celkovo má model MobileNet 30 vrstiev[11].

3.3.3 Trénovanie

Tento model bol trénovaný pomocou asynchrónneho gradientného zostupu. Trénovanie zjednodušoval fakt, že vo všeobecnosti sú malé modely menej náchylné na preučenie[11].



Obr. 3.4: Rozloženie klasickej konvolúcie na hĺbkovo oddeliteľnú

Kapitola 4

Výpočtové prostredie

Napriek tomu že funkčné algoritmy hlbokého učenia sú známe od osemdesiatych rokov minulého storočia, boli považované za príliš výpočetne náročné na to, aby bol možný veľký výskum s hardvérom dostupným pred rokom 2006[12]. Konvolučné operácie sú výpočetne náročné, a tak robia DCNN signifikantne pomalšie na vyhodnotenie v porovnaní so štandartnými ANN. Na prekonanie týchto obmedzení boli navrhnuté tri metódy na zrýchlenie DCNN: rozbalenie konvolúcie, použitie softvéru na výpočty v lineárnej algebre a využitie GPUs[1].

4.1 Google Colaboratory

Koncom roka 2017 predstavila spoločnosť Google projekt nazvaný Google Colaboratory. Umožňuje používateľom pracovať na Jupyter Notebookoch priamo prostredníctvom internetového prehliadača[13]. Toto prostredie podporuje populárne knižnice ako napríklad Keras, TensorFlow, PyTorch alebo OpenCV a so súbormi sa dá pracovať pomocou bash príkazov. Najväčšou výhodou tohto projektu je možnosť zadarmo používať na výpočty grafické karty od spoločnosti NVIDIA - Tesla K80[14].

Na ďalšie experimenty som sa rozhodol použiť 32 konvolučných filtrov s receptorovým poľom o rozmeroch 3×3 pixely s posunom jeden pixel cez sto náhodných obrázkov o veľkosti 244×244 pixelov. Pri desiatich spusteniach bol výpočet na GPU 62 krát rýchlejší oproti CPU, viď tabuľku 4.1.

Hlavná výhoda použitia tohto prostredia oproti klasickým zdieľaným výpočetným uzlom (napr. MetaCentrum) je okamžité spustenie programu a dá sa okamžite sledovať stratovosť modelu a tak aj experimentovať pri prvotnom kontakte s modelmi s rôznymi nastaveniami stupňa učenia a sledovaním vývoja presnosti modelu. Najväčšou nevýhodou je potreba neustálej prezencie v aplikácii počas dlhších výpočtov ako aj faktu, že pri opätovnom použití prostredia je človek pripojený na iný výpočtový uzol, takže je potrebné si dáta zálohovať. Taktiež pri trénovaní s veľkým objemom dát je potrebné si dáta

vždy skopírovať do výpočtového prostredia, pretože aj pri možnosti napojenia sa na Google Drive, trénovanie za použitia dát priamo z tejto aplikácie kôli ich prenosu značne predlžuje dobu behu.

Figúra 4.1 ukazuje prácu v Google Colaboratory. Konkrétne stiahnutie modelu ResNet50 s váhami predtrénovanými na ImageNat a ukážka spúšťania bash príkazov, ktoré sa spúšťajú pomocou znaku "!" na začiatku riadku.

HW	Čas [s]
CPU	59.0536
GPU	0.9425

Tabuľka 4.1: Porovnanie doby behu na CPU versus GPU

```
from keras.applications.resnet50 import ResNet50
!git clone https://github.com/kostelansky17/bap.git
resnet50 = ResNet50(weights='imagenet')
```

Obr. 4.1: Príklad práce v Goole Colaboratory

Kapitola 5

Robot NAO

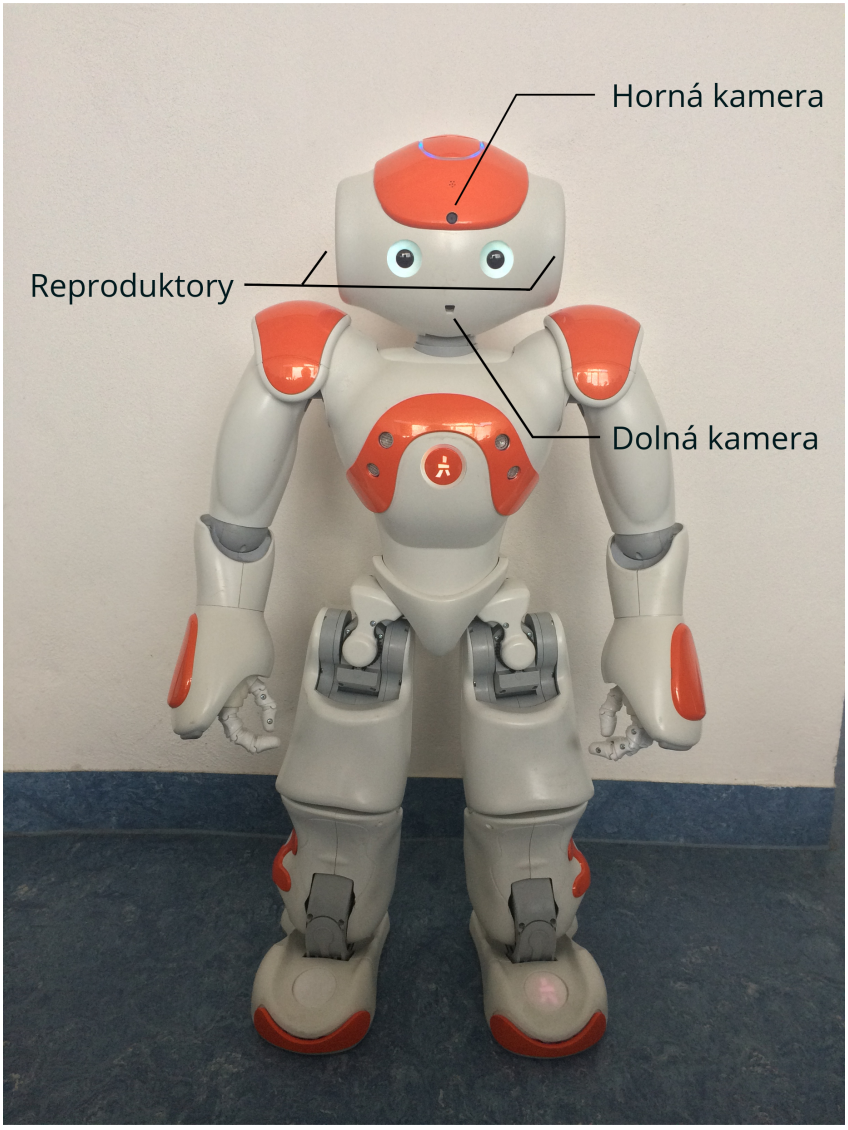
Robot NAO používaný v tejto práci je humanoidná robotická platforma vyrábaná spoločnosťou Aldebaran Robotics. Je využívaný najmä vo vzdelávaní a výskume. NAO je približne 58 centimetrov vysoký, schopný pohybu a rozprávania. Obsahuje množstvo senzorov a výpočtovú jednotku s parametrami 1,6GHz CPU, 1GB RAM, 2GB rýchlej pamäte a 8GB micro SDHC. Pripojenie je možné ethernetovým káblom a prostredníctvom WiFi. Najdôležitejšie senzory pre túto prácu sú dve kamery umiestnené na hlave. Je vhodné zmieniť, že kamery sa nenachádzajú priamo v očiach, ale jedna na čele a druhá "v oblasti úst", viď figúra 5.1. Tie sú schopné poskyktnúť rozlíšenie až 1280×960 pixelov pri 30 snímkach za sekundu. Zorné pole každej kamery je $60,97$ stupňov horizonálne a $47,64$ stupňov vertikálne[15]. Ďalej sú pre moju prácu podstatné dva reproduktory umiestnené v oblasti uší, pomocou ktorých bude NAO komunikovať s okolím.

5.1 NAOqi framework

NAOqi je framework vyvinutý spoločnosťou Aldebran Robotics, a bol nasadený spolu s robotom NAO. Je to distribúcia Linuxu založená na Gentoo a slúži ako operačný systém pre robota a umožnuje multiplatformný vývoj programov na NAO. Hlavnými programovacími jazykmi sú Python a C++. V mojej práci používam verziu 2.1.4.13. NAOqi API je rozdelené na niekoľko častí, kde každá umožňuje prístup k odlišným funkciám a systémom na robotovi[16]:

- NAOqi Core obsahuje hlavné moduly pre prácu s robotom, ktoré sú vždy dostupné.
- NAOqi Vision je časť API, ktorá je v mojej práci použitá na zachytenie fotografií.
- NAOqi Audio slúži na prácu so zvukovým softvérom. Pomocou funkcií v tejto časti API komunikuje NAO s okolím.

- NAOqi Motion je hlavný nástroj umožnujúci robotovi pohyb. Metódy použité v mojej práci slúžia na vstávanie a pohyb hlavou.



Obr. 5.1: Robot NAO

Kapitola 6

Keras

Keras je open-source knižnica, slúžiaca na vytváranie neurónových sietí v jazyku Python. Je schopný pracovať nad TensorFlow, Theano alebo CNTK. Bol vytvorený tak, aby umožnil rýchle experimentovanie s hlbokými neurónovými sieťami[17]. Pri svojej práci som pužíval Keras 2.1.6. Počas trénovania sietí som ako backend používal TensorFlow vo verzii 1.7.0. Počas vývoja aplikácie sa mi však použitím backendu Theano 1.0.1 podarilo zabezpečiť väčšiu stabilitu behu. Figúra 6.1 znázorňuje prácu s frameworkom Keras. Konkrétne stiahnutie modelu ResNet50 s váhami predtrénovanými na datasete ImageNet, predspracovanie obrázku určeného na klasifikáciu a následná klasifikácia s výpisom troch najpravdepodobnejších karegórií.

```
from keras.applications.resnet50 import ResNet50
from keras.applications.resnet50 import preprocess_input
from keras.applications.resnet50 import decode_predictions
from keras.preprocessing import image
import numpy as np

model = ResNet50(weights='imagenet')

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

Obr. 6.1: Príklad použitia frameworku Keras[6]

Kapitola 7

Testovanie a trénovanie vybraných modelov

7.1 Testovanie predtrénovaných modelov

Tri vybrané modely s váhami predtrénovanými na datasete ImageNet som testoval na mnou vytvorenom sete fotografií pozostávajúcom z predmetov denného použitia (napr. kuchynské náčinie, ovocie, knihy, náramkové hodinky, atď). Niektotré predmety boli fotografované z rôznych uhlov, ľahších aj ťažších na klasifikáciu. Celkovo dataset obsahoval 50 fotografií. Hodnotil som ich klasifikačnú presnosť z hľadiska Top-1 a Top-5 klasifikácie, podobne ako ILSVRC súťaž. Najpresnejší bol podľa očakávaí Resnet50, za ním VGG16 a najmenej presný bol model MobileNet. Presné výsledky sú uvedené v tabuľke 7.1.

Model	Top-1	Top-5
VGG16	0.36	0.72
ResNet50	0.56	0.9
MobileNet	0.08	0.16

Tabuľka 7.1: Výsledky testovania predtrénovaných modelov

7.2 Testovanie presnosti modelov vzhľadom na hĺbku dotrénovania

Modely ResNet50 a VGG16 som testoval na klasifikácii kvetov. Testovacie dáta obsahovali fotografie piatich typov kvetov: sedmokráska, púpava, ruža, slnečnica a tulipán. V mojich pokusoch som skúmal vplyv hĺbky a dĺžku trénovania na presnosť klasifikácie.

7.2.1 Trénovanie

Dáta som rozdelil na dve množiny, trénovaciu po 500 fotiek každého kvetu a validačnú obsahujúcu 100 fotografií každého druhu. Modely som trénoval pomocou SGD s momentom. Stupeň učenia bol nastavený na 10×-3 a postupne sa znižoval ako sa ustáľovala presnosť. Celkovo sa stupeň učenia znížil dva krát, a to tak, že bol vydelený 10. Momentum bolo nastavené na $0,9$ ako pri trénovaní na dátach ImageNet. Fotografie som rozdelil do 25 skupín a celkovo testovanie prebiehalo 100 epôch. Figúra 7.1 ukazuje trénovanie modelu vo frameworku Kertas.

```
model.compile(optimizer=SGD(lr=0.001, momentum=0.9), loss='
                                categorical_crossentropy')
model.fit_generator(train,epochs = 10, steps_per_epoch = 25,
                                validation_data=valid)
```

Obr. 7.1: Trénovanie modelu vo frameworku Keras

7.2.2 VGG16

Pôvodné plne spojité vrstvy som nahradil novými s náhodne inicializovanými váhami. Najskôr som skúšal dotrénovať len tieto vrstvy, neskôr som vždy ladil aj predtrénované váhy v konvolučných vrstvách, po jednotlivých blokoch, na obrázku 3.1 sú tieto bloky rozlíšené farebne. Vytvorenie tohto modelu popisuje figúra 7.2.

```
vgg16 = VGG16(weights = 'imagenet',include_top = True)
x = Desne(4096, activation = 'relu')(vgg16.layers[-5].output)
x = Dense(4096, activation = 'relu')(x)
x = Dense(5, activation = 'softmax')(x)
model = Model(inputs = vgg16.input, outputs = x)
```

Obr. 7.2: Vytvorenie použitého modelu VGG16 vo frameworku Keras

7.2.3 ResNet50

Pri tomto modeli som tiež nahradil pôvodú plne spojitú vrstvu novou a skúšal ju dotrénovať. Následne som skúšal dotrénovávať bloky CONVL, podobne ako pri VGG16. Na figúre 3.3 je vidieť jednotlivé bloky oddelené farebne. Vytvorenie modelu ResNet50 je znázornené v ukážke 7.3.

```
resnet50 = ResNet50(weights='imagenet', include_top=True)
x = Dense(5, activation='softmax')(resnet50.layers[-2].output)
model = Model(inputs=resnet50.input, outputs=x)
```

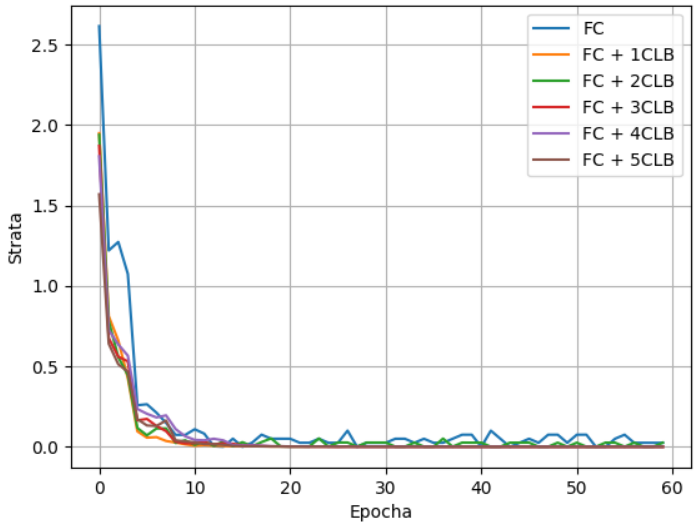
Obr. 7.3: Vytvorenie použitého modelu ResNet50 vo frameworku Keras

7.2.4 Výsledky

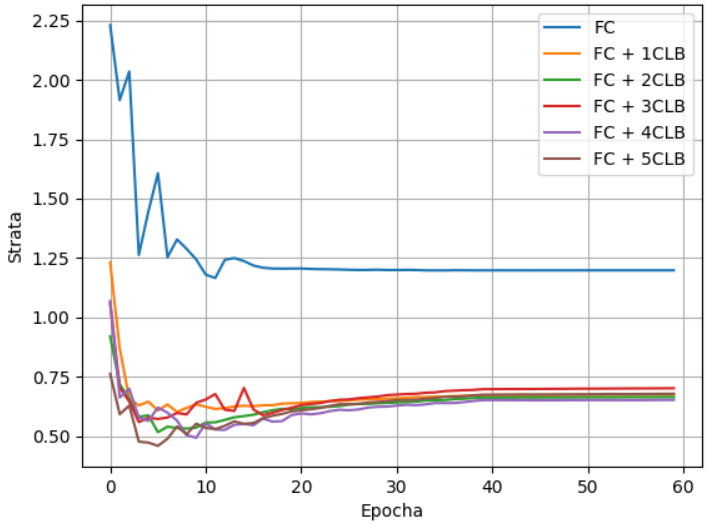
Model VGG16 pri každej konfigurácii konvergoval k optimu rýchlo, majoritne v prvých desiatich epochách. Následne len osciloval okolo neho, viď graf 7.4. Ako je vidno na grafe 7.5, doučovanie len jedného bloku konvolučných vrstiev malo veľký vplyv na klasifikačnú presnosť. So zvyšujúcou hĺbkou ladenia sa validačná strata výrazne nemenila.

Pri modele ResNet50 konvergencia k optimu trvala dlhšie, približne 40 epôch. Na grafoch 7.6 a 7.7 môžeme vidieť úmerné znižovanie trénovacej a validačnej straty s narastajúcou hĺbkou, od ktorej boli konvolučné vrstvy doladované.

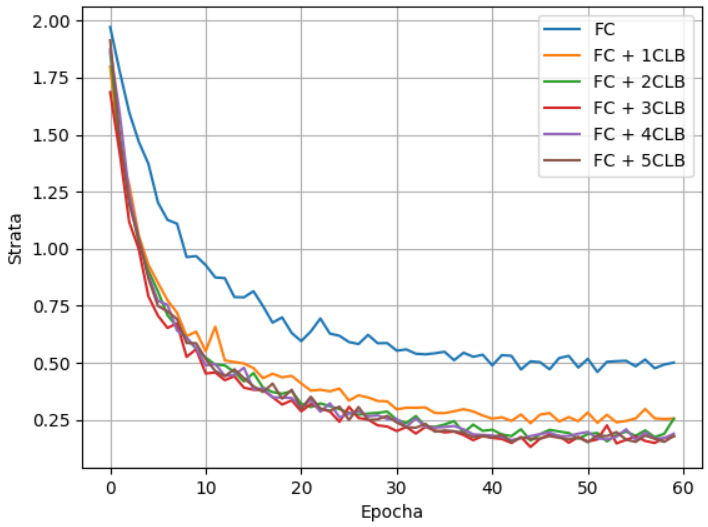
Na grafe 7.8 je porovnanie najúspešnejších verzií oboch typov modelov. Boli to modely, v ktorých boli trénované všetky vrstvy. Je vidieť, že napriek tomu, že VGG16 má vyššiu schopnosť znížiť stratu na trénovacích dátach, ResNet50 je úspešnejší na validačnej množine, a dá sa povedať že je vidieť odolnosť reziduálneho učenia na problém preučenia.



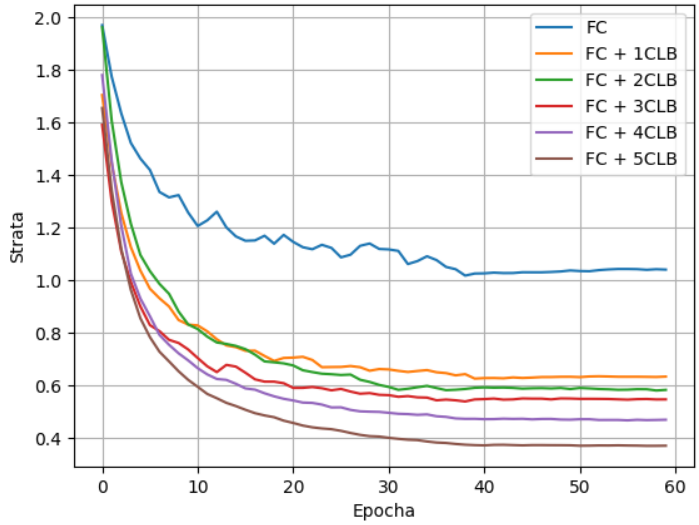
Obr. 7.4: Trénovacia strata VGG16 v priebehu trénovania



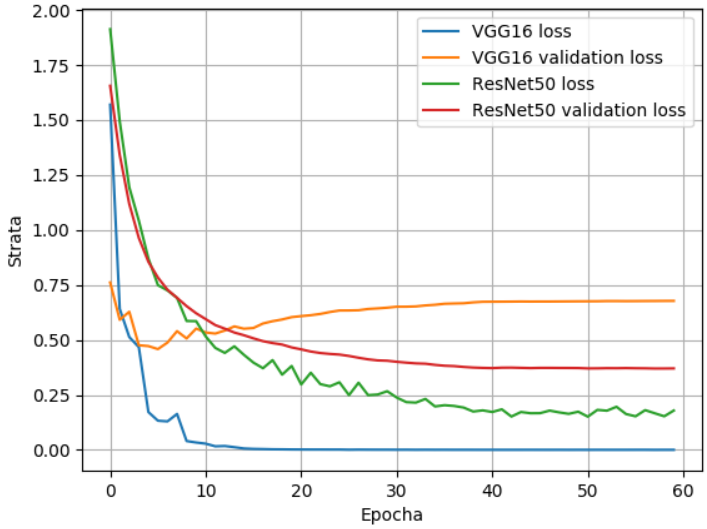
Obr. 7.5: Validačná strata VGG16 v priebehu trénovania



Obr. 7.6: Trénovacia strata ResNet50 v priebehu trénovania



Obr. 7.7: Validačná strata ResNet50 v priebehu trénovania



Obr. 7.8: Porovnanie VGG16 a ResNet50

7.3 Trénovanie na vytvorenom datasete školy

V tejto časti popisujem trénovanie vybraných konvolúčných sietí metódami vyplývajúcimi z predchádzajúcich experimentov. Modely som dotrénoval na vlastnom datasete Fakulty informačných technológií a najlepší z modelov bol použitý pri aplikácii na robota NAO.

7.3.1 Dáta

Vytvoril som dataset obsahujúci fotografie tried, chodieb, schodísk a priestorov pri výťahoch v budovách TH:A a T9 Fakulty informačných technológií ČVUT v Prahe. Celkovo obsahuje 2774 fotiek. Fotky sú z rôznych uhlov a za rôzneho osvetlenia. Dáta boli kategorizované dvoma spôsobmi, buď podľa miesta z akého pochádzajú, alebo podľa toho v ktorej budove boli nafotené. Počet fotiek v jednotlivých kategóriách je uvedený v tabuľke 7.2. Datasety boli rozdelené na trénovaciu a validačnú množinu. Validačná bola vytvorená náhodným výberom z každej kategórie. Presné veľkosti trénovacích a validačných množín popisujú tabuľky 7.3 a 7.4.

Kategória	TH:A	T9
Učebňa	545	626
Hala	319	392
Výťah	90	113
Schodisko	185	504

Tabuľka 7.2: Počet fotografií v jednotlivých kategóriách

Kategória	Trénovacia	Validačná
Učebňa	1071	100
Hala	651	60
Výťah	183	20
Schodisko	639	50

Tabuľka 7.3: Počet fotografií v trénovacej a validačnej množine datasetu priestorov.

Kategória	Trénovacia	Validačná
TH:A	1039	100
T9	1505	130

Tabuľka 7.4: Počet fotografií v trénovacej a validačnej množine datasetu budov.

7.3.2 Trénovanie

Na trénovanie som použil algoritmus SGD s momentom. Trénovacie dáta boli v každej iterácii rozdelené na 30 menších skupín a momentum bolo nastavené na 0.9. Trénovanie prebiahalo 60 epôch a stupeň učenia bol znížený dva krát vydelením desiatimi ako sa zmierňovalo učenie v priebehu iterácií.

Zároveň som vytvoril konvolučnú neurónovú sieť, ktorá vychádza z DCNN ResNet50, ale namiesto jednej plne spojitej vrstvy obsahuje tri, ako sieť VGG16. Táto DCNN bude nazývaná ResNet50_B, klasická sieť ResNet50 s jednou plne spojitou vrstvou bude nazývaná ResNet50_A. Vytvorenie modelu ResNet50_B je ukázané vo figúre 7.9. V týchto dvoch sieťach boli vytvorené nové plne spojité vrstvy s náhodne inicializovanými váhami a konvolučné vrstvy s váhami predtrénovanými datasete ImageNet. V sieťach založených na architektúre ResNet50 boli vzhľadom na predchádzajúce experimenty dotrénované celé a stupeň učenia začínal na 10^{-3}.

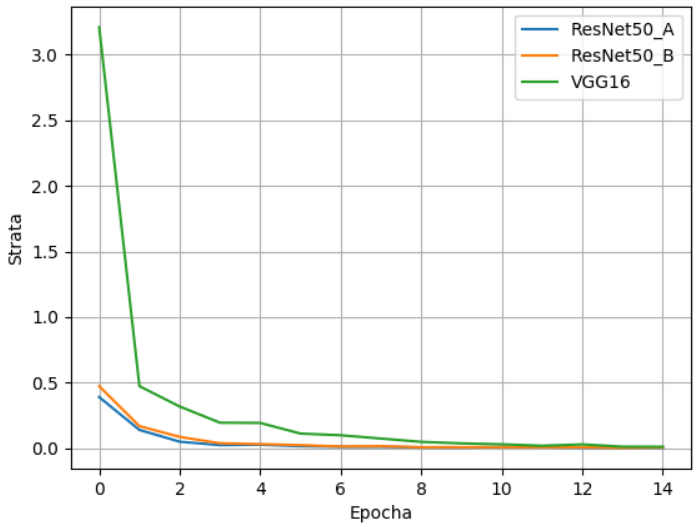
Pri DCNN VGG16 som vytvoril nové plne spojité vrstvy s náhodne inicializovanými váhami, a na základe prechádzajúcich výsledkov boli dotrénované plne spojité vrstvy a dva najvrchnejšie bloky konvolučných vrstiev. Prvotný stupeň učenia bol nastavený na 10^{-4}.

```
resnet50 = ResNet50(weights = 'imagenet', include_top = True)
x = Dense(4096,activation = 'relu')(resnet50.layers[-2].output)
x = Dense(4096, activation = 'relu')(x)
x = Dense(4, activation='softmax')(x)
model = Model(inputs=resnet50.input, outputs=x)
```

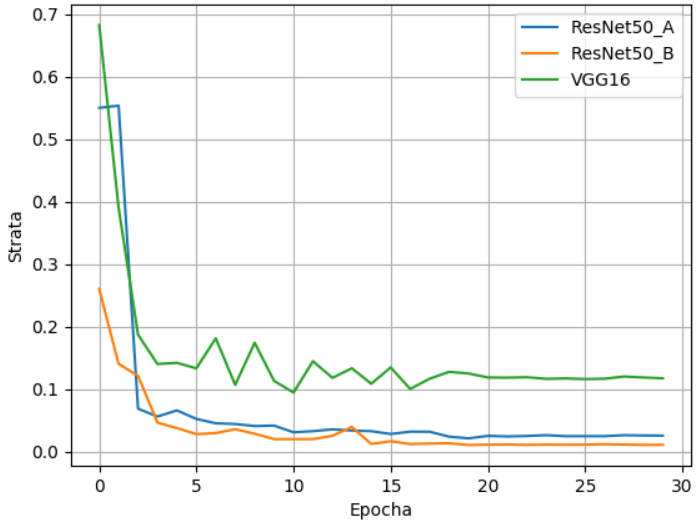
Obr. 7.9: Vytvorenie použitého modelu ResNet50_B vo frameworku Keras

7.3.3 Výsledky

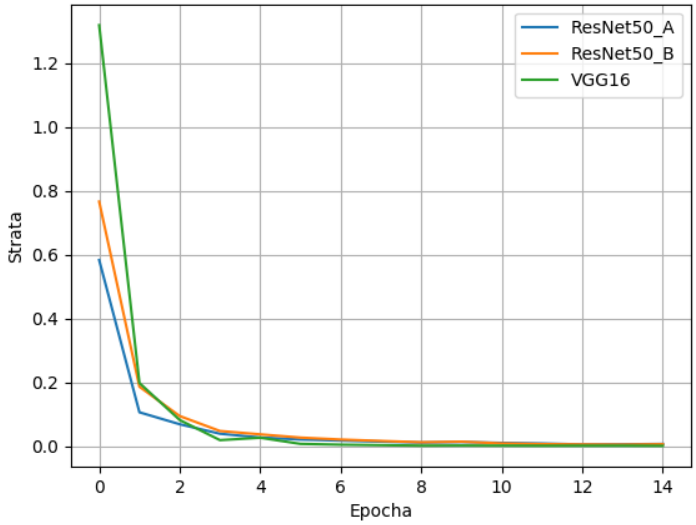
Ako je vidieť na grafoch 7.10 a 7.12 tak všetky tri modely konvergovali k lokálnym optimám už v prvých epocháh. Model VGG16 mal spočiatku najvyššiu stratu ale konvergoval porovnateľne s modelmi vytvorenými na základe architektúry ResNet. Na grafoch 7.11 a 7.13 je porovnanie straty na validačnom datasete pri oboch klasifikáciách. Ako je vidieť, siete na báze ResNet boli opäť pri klasifikácii úspešejšie. Najpresnejšia bola sieť ResNet50_B. Ukázalo sa, že pridanie plne spojitých vrstiev, má za účelom zvýšenie presnosti a toto prehĺbenie siete nemá vplyv na možné preučenie.



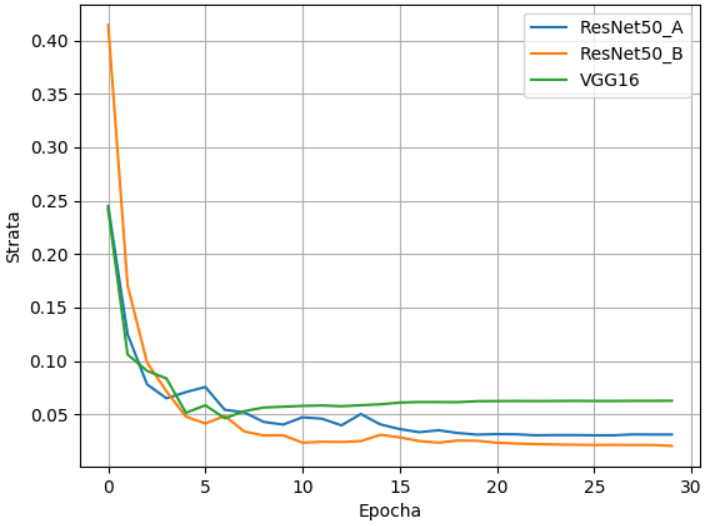
Obr. 7.10: Trénovacia strata pri klasifikácii budov



Obr. 7.11: Validačná strata pri klasifikácii budov



Obr. 7.12: Trénovacia strata pri klasifikácii priestorov



Obr. 7.13: Validačná strata pri klasifikácii priestorov

Kapitola 8

Implementácia

V tejto kapitole popisujem postup pri vytvorení aplikácie na robota NAO. V aplikácii robot dokáže opakovane klasifikovať miestonsť a budovu v ktorej sa nachádza na základe záberov zachytených jeho kamerou. Pri vývoji bol použitý Python 2.7 s Kerasom 2.1.6 bežiacom na Theano 1.0.1 backende na operačnom systéme Ubuntu 17.10. Ďalším potrebným komponentom bol framework NAOqi vo verzii 2.1.4.13 a knižnica OpenCV. Všetky nasledujúce procesy niesú spúšťané priamo na robotovi NAO ale na počítači, ktorý je k nemu pripojený a komunikuje s ním pomocou frameworku NAOqi. Toto riešenie som zvolil kôli výpočtovej náročnosti práce s CNNs, a faktom že na robota nieje možné doinštalovať softvér potrebný na prácu s modelmi.

8.1 Výber modelov

Najpresnejším z trénovaných modelov v oboch kategiách bol ResNet50_B. Avšak pri výslednom použití je potrebné zvážiť aj iné faktory ako len presnosť klasifikácie. Keďže sa jedná o interaktívnu aplikáciu, dôležitá je aj rýchlosť hodnotenia a čas za aký je aplikácia pripravená na použitie. Pred samotným použitím, je potrebné načítať oba modely, kde hrá kľúčovú úlohu ich veľkosť. Tabuľka 8.1 porovnáva veľkosti modelov a rýchlosť načítania. Aj keď medzi veľkosťami modelov ResNet50_A a ResNet50_B je signifikantný rozdiel, modely je potrebné načítať len raz, pri spustení aplikácie. Rýchlosť klasifikácie má po niekoľkých použitiach tendenciu výrazne klesať a veľkosť modelu už tak nehrá rolu. Keďže je pre túto aplikáciu kľúčová práve presnosť, rozhodol som sa použiť model ResNet50_B, aj napriek dlhšej dobe načítania pri spúšťaní.

Model	Veľkosť [MB]
ResNet_A	180
ResNet_B	373
VGG16	1018

Tabuľka 8.1: Porovnanie veľkostí modelov

8.2 Pohyb a odfotenie priestoru

Pri spustení aplikácie sa NAO postaví do vzpriamenej polohy. Pri zachytávaní jednotlivých snímok robot natáča hlavu do polôh, v ktorých sú zostrojené fotografie. Celkovo sú vytvorené tri fotografie v poloháčh $[0, 0]$, $[0.7, -0.3]$, $[-0.7, -0.3]$. Uhly sú zadávané v radiánoch. Pred použitím kamier je potrebné prihlásiť sa ku kamere a po prevedení fotografií sa odhlásiť. V mojej práci požívam hornú kameru, rozlíšenie 640×480 pixelov a BGE ColorSpace. Ukážku použitia presných metód znázorňuje figúra 8.1.

8.3 Klasifikácia

Všetky zachytené fotografie sú jednotlivo ohodnotené modelmi pre obe požadované kategórie. Tieto výsledky sú ďalej spracované, kedže NAO pri pozorovaní priestoru otáča hlavou, je možné že v priestoroch pri výťahu sú niektoré fotografie ohodnotené ako chodba a len jedna ako výťah. V celkovom súčte, by tak napokon chodba zvíťazila nad výťahom, čo je ale požadovaná lokalita. Preto pri klasifikácii zvažuje, či sa na jednej z fotografí nenachádza s vysokou pravdepodobnosťou zložitý priestor (trieda, výťah, schodisko) a na ostatných chodba. Ak táto situácia nastala, NAO priestor klasifikuje ako ten zložitejší. V opačnom prípade nastane súčet všetkých hodnotení. Ten taktiež nastáva pri klasifikácii budov. Na základe tejto predikcie robot určí svoju polohu s označením stupňa istoty o prítomnosti v danom priestore, a to slovne frázami "Som si isty ze som", "Asi som" a "Tipol by som si ze som v".

8.4 Interakcia

NAO komunikuje s okolím počas celého behu aplikácie, a naopak používateľ interaguje s robotom pomocou tlačidiel na jeho hlave. Stredným, spustí klasifikáciu a pomocou predného tlačidla sa NAO postaví do vzpriamenej polohy. Použitie NAOqi na zachytenie dotykov na tlačidlách umiestnených na hlave a hovorenie pomocou ALTextToSpeech je ukázané vo figúre 8.2.

```
# vytvorenie proxy - pripojenie k~robotovi
posture = naoqi.ALProxy("ALRobotPosture", IP, 9559)
motion = naoqi.ALProxy("ALMotion", IP, 9559)
video = naoqi.ALProxy("ALVideoDevice", IP, 9559)

# pozicie do ktorych bude NAO otacat hlavu
headPositions = [[0,0],[0.7,-0.2],[-0.7,-0.1]]

# postavenie robota do vzpriamenej polohy
posture.goToPosture("Stand", 1)

name = "cam"
ID = 0
resolution = 2
colorSpace = 13
fps = 30
# zaregistrovanie kamery
cam = video.subscribeCamera(name,ID,resolution,colorSpace,fps)

for i in range(0,3):
  # nastavenie klbov do daneho uhla
  motion.setAngles("HeadYaw",headPositions[i][0], 0.15)
  motion.setAngles("HeadPitch",headPositions[i][1], 0.15)

  # obdrzanie najnovsej fotografie z~kamery
  image = video.getImageRemote(cam)

  # spracovanie fotografie
  im = image[6]
  nparr = np.fromstring(im, np.uint8)
  nparr = nparr.reshape(480, 640, 3)
  cv2.imwrite(images[i], nparr)

# odregistrovanie kamery
video.unsubscribe(cam)
```

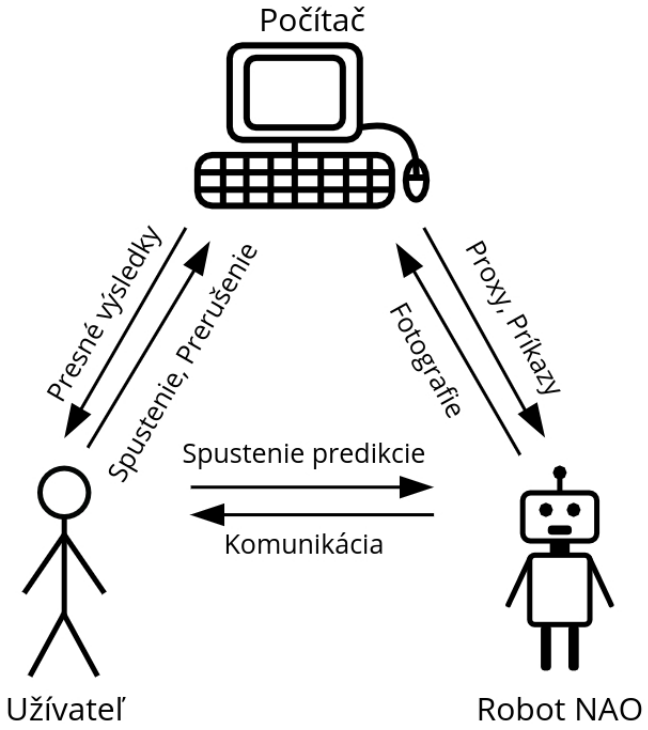
Obr. 8.1: Zachytenie fotografií a ovládanie pohybu cez NAOqi API[18]

```
class ReactToTouch(naoqi.ALModule):
    def __init__(self):
        naoqi.ALModule.__init__(self, self.__class__.__name__)
        # vytvorenie proxy - pripojenie k~robotovi
        self.tts = naoqi.ALProxy("ALTextToSpeech")
        self.memory = naoqi.ALProxy("ALMemory")

    # prihlasenie k~udalosti, ktore automaticky spusti metodu
        self.memory.subscribeToEvent("FrontTactilTouched",
                        self.__class__.__name__, "onFront")
        self.memory.subscribeToEvent("MiddleTactilTouched",
                        self.__class__.__name__, "onMiddle")
        self.memory.subscribeToEvent("RearTactilTouched",
                        self.__class__.__name__, "onRear")
    def onFront(self, event_name, value):
      if value == 1:
        # povedanie specifikovaneho retazca
        self.tts.say("Stlacil si vpredu!")
    def onMiddle(self, event_name, value):
      if value == 1:
        # povedanie specifikovaneho retazca
        self.tts.say("Stlacil si vstrede!")
    def onRear(self, event_name, value):
      if value == 1:
        # povedanie specifikovaneho retazca
        self.tts.say("Stlacil si vzadu!")

# vytvorenie vlastneho brokera
myBroker = naoqi.ALBroker("myBroker", "0.0.0.0", 0, IP, 9559)

# premenna je vzdy global a vola sa rovnako ako trieda
global ReactToTouch
ReactToTouch = ReactToTouch()

try:
    import time
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    myBroker.shutdown()
```

Obr. 8.2: Použitie NAOqi na komunikáciu a reagovanie na stlačenie tlačidiel[18]

8.5 Architektúra

Užívateľ pomocou počítača pripojeného k robotovi NAO spúšťa a ukončuje aplikáciu. Počítač komunikuje s robotom, dostáva od neho fotografie, ktoré hodnotí a výstupy klasifikácie mu posiela naspäť formou príkazov slúžiacich na komunikáciu s okolím. Na počítači sa taktiež zobrazujú podrobné hodnotenia každého klasifikovaného pohľadu, ako aj celkový výsledok. Užívateľ stlačením stredného tlačidla na robotovej hlave dá podnet na spustenie klasifikácie. Detailne tieto vzťahy popisuje obrázok 8.3.



Obr. 8.3: Architektúra aplikácie pre robota NAO

8.6 Príklad použitia

Po zapnutí robota a pripojení k nemu sa aplikácia spustí z príkazového riadku. Ako parameter je potrebné zadať IP adresu robota NAO, ukážka 8.4. Po spustení NAO používateľa pozdraví a požiada ho o chvíľu strpenia, pokiaľ sa načítajú modely. Keď je všetko pripravené, robot ohlási, že je pripravený na klasifikáciu. Po stlačení stredného tlačidla sa NAO rozhliadne, a po chvíli oznámi kde si myslí že sa nachádza. V prípade, že je potrebné aby sa NAO opako-

vane postavil, stačí stlačiť predné tlačidlo. Aplikácia sa ukončuje z terminálu v ktorom je spustená prerušením klávesovou kombináciou "Ctrl + C".

```
python nao_app.py 169.254.129.144
```

Obr. 8.4: Spustenie aplikácie pre robota NAO

Kapitola 9

Výsledky a diskusia

Výslednú aplikáciu som testoval v priestoroch školy, na ktoré boli siete trénované. Pri testoch bol NAO umiestňovaný rozumným smerom s výhľadom. Otáčanie ho napríklad k stene v triede, by malo za výsledok pomýlenie sa za chodbu, pretože to je priestor pozostávajúci hlavne zo stien, podláh bez predmetov. Vytvoril som matice zámien pre miesta, a pre budovy. Ako je vidieť na výsledkoch v tabuľke 9.1, pri predikcii miesta dosahuje NAO veľmi vysoké výsledky. Presnosť klasifikácie na tejto kategórii bola $93,48\%$. Na ďalšej tabuľke 9.2 je vidieť, že robot mal výraznejšie problémy pri klasifikovaní budovy, mal sklon predikovať budovu T9. Je to spôsobné najmä veľkými podobnosťami medzi triedami, tie sa líšia len v detailoch, ktoré NAO nebol vždy schopný odhaliť. Mohlo to byť tiež spôsobené zlými svetelnými podmienkami v budove TH:A, ako aj väčším počtom fotografií z budovy T9 v trénovacom datasete. Celkovo mal NAO pre budovy presnosť $76,09\%$. Tabuľka 9.3 obsahuje presnosť, senzitivitu a špecifickosť pre každú kategóriu.

		Skutočnosť			
		Trieda	Výťah	Chodba	Schody
Predikcia	Trieda	15	1	0	2
	Výťah	0	8	0	0
	Chodba	0	0	12	0
	Schody	0	0	0	8

Tabuľka 9.1: Výsledky testovania pre miesta

		Skutočnosť	
		T9	TH:A
Predikcia	T9	23	11
	TH:A	0	12

Tabuľka 9.2: Výsledky testovania pre budovy

Kategória	Presnosť	Senzitivita	Špecifickosť
Trieda	0,9348	1	0,8909
Výťah	0,9783	0,8889	1
Chodba	1	1	1
Schody	0,9565	0,8	1
T9	0,7609	1	0,5217
TH:A	0,7609	0,5217	1

Tabuľka 9.3: Výsledky pre všetky kategórie

Záver

V práci som sa zaoberal architektúrou hlbokých konvolučných neurónových sietí a ich praktickými implementáciami. Následne som skúmal možnosti ich dotrénovania na nové kategórie, ako aj vplyv hĺbky dotrénovania na klasifikačnú presnosť. Vytvoril som dataset priestorov Fakulty informačných technológií ČVUT v Prahe pozostávajúci z takmer tritisíc fotogragií. Počas experimetov som vytvoril novú konvolučnú neurónovú sieť pridaním plne spojitých vrstiev na existujúcu sieť ResNet50 a ukázalo sa, že pridanie týchto vrstiev zvyšuje klasifikačnú presnosť, a zároveň je odolné voči preučeniu. Spôsobmi, ktoré vyplynuli z mojich pozorovaní som skúmané konvolučné siete dotrénoval na vytvorenom datasete fakulty. Vybral som najvhodnejšie modely na implementáciu na robota NAO a vytvoril som aplikáciu v ktorej NAO klasifikuje miestnosť a budovu kde sa práve nachádza. Úspešnosť finálnej aplikácie pri klasifikácii preistorov ukázala veľmi vysokú presnosť, mala však prblémy pri predikcii budovy. Ukázal som, že modely predtrénované na datasete ImageNet sa dajú dotrénovať pre nové kategórie s vysokou úspešnosťou klasifikácie. Načrtol som široké možnosti využitia konvolučných neurónových sietí na robotoch.

V budúcnosti by bolo možné vytrénované modely použiť v rôznych iných inteligentných aplikáciach ako ich aj dotrénovať na obšírnejšom datasete fakulty ďalšími priestormi a kategóriami. Konvolučné neurónové siete majú veľa možností ďalšieho využitia nielen na robota NAO, ale za úvahu určite stojí preskúmanie možností ich implementácie na robota Pepper[19].

Literatúra

[1] Rawat, W.; Wang, Z.: Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation*, ročník 29, č. 9, 2017: s. 2352–2449, doi:10.1162/neco_a_00990, pMID: 28599112. Dostupné z: https://doi.org/10.1162/neco_a_00990

[2] Nielsen, M.: Neural Networks and Deep Learning [online]. dec 2017, [cit. 7. 5. 2018]. Dostupné z: http://neuralnetworksanddeeplearning.com/chap6.html

[3] ReLU and soft-max activation functions [online]. feb 2017, [cit. 8. 5. 2018]. Dostupné z: https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions

[4] Ng, A.: Optimization: Stochastic Gradient Descent [online]. [cit. 7. 5. 2018]. Dostupné z: http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/

[5] ImageNet stats [online]. apr 2010, [cit. 7. 5. 2018]. Dostupné z: http://www.image-net.org/about-stats

[6] Keras Applications [online]. [cit. 7. 5. 2018]. Dostupné z: https://keras.io/applications/

[7] Simonyan, K.; Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*, sep 2014: s. 1–5, 1409.1556.

[8] ILSVRC 2014 results font[online]. [cit. 7. 5. 2018]. Dostupné z: http://www.image-net.org/challenges/LSVRC/2014/results

[9] ILSVRC 2015 results [online]. [cit. 7. 5. 2018]. Dostupné z: http://www.image-net.org/challenges/LSVRC/2015/results

[10] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. *ArXiv e-prints*, dec 2015: s. 1–5, 1512.03385.

[11] Howard, A. G.; Zhu, M.; Chen, B.; aj.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv e-prints*, apr 2017: s. 1–4, 1704.04861.

[12] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press, 2016. Dostupné z: http://www.deeplearningbook.org

[13] Howal, S.: Neural Networks with Google CoLaboratory [online]. dec 2017, [cit. 7. 5. 2018]. Dostupné z: https://towardsdatascience.com/neural-networks-with-google-colaboratory-artificial-intelligence-getting-started-713b5eb07f14

[14] Google Colab Free GPU Tutorial [online]. jan 2018, [cit. 7. 5. 2018]. Dostupné z: https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d

[15] NAO - Technical overview [online]. [cit. 8. 5. 2018]. Dostupné z: http://doc.aldebaran.com/2-1/nao/index.html

[16] NAOqi API [online]. [cit. 8. 5. 2018]. Dostupné z: http://doc.aldebaran.com/2-1/naoqi/index.html

[17] Keras [online]. [cit. 8. 5. 2018]. Dostupné z: https://keras.io/

[18] Edux FIT ČVUT BI-ZIVS [online]. feb 2018, [cit. 8. 5. 2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-ZIVS/labs/python-examples

[19] Robot Pepper [online]. [cit. 8. 5. 2018]. Dostupné z: https://www.ald.softbankrobotics.com/en/robots/pepper/find-out-more-about-pepper

Dodatok **A**

Zoznam použitých skratiek

CNN(s) Konvolučná neurónová sieť (siete)

DCNN(s) Hlboká konvolučná neurónová sieť (siete)

ANN Umelá neurónová sieť

MAXPOOL Vrstva maximálneho spájania

AVGPOOL Vrstva priemerného spájania

FC Plne spojitá vrstva

XCLB X posledných blokov konvolučných vrstiev

SGD Stochastický gradientný zostup

HW Hardvér

CPU Centrálna procesorová jednotka

GPU(s) Grafický procesor (procesory)

Dodatok **B**

Obsah priloženého DVD

- `readme.txt` ..... stručný popis obsahu DVD
- `exe` ..... adresár so spustiteľnou formou implementácie
  - `scripts` ..... skripty vytvárajúce modely
  - `app` ..... aplikácia pre robota NAO
- `models` ..... predtrénované modely
- `thesis` ..... zdrojová forma práce vo formáte LaTeX
  - `src` ..... prílohy
- `text` ..... text práce
  - `BP_Kostelansky_Martin.pdf` ..... text práce vo formáte PDF